Magma Pygments

'Immir'

March 2023

1 Introduction

This is a demo of a simple Pygments [1] lexer for the language of the computer algebra system Magma [2]. Together with the LaTeX `minted` [3] package, this allows syntax-highlighted Magma source code to typeset in documents.

In the Appendix one can find the Python source for this lexer – formatted using the standard Pygments lexer for Python for comparison with the Magma output in the examples following.

At some point a `git` repo will likely be published for anyone wishing to contribute. For the moment, anyone who knows *Immir* knows how to contact him...

2 Some Magma examples

Some intrinsics from a small Magma package:

```
// file: intrin.m

intrinsic '@' (x, f::RngUPolElt) -> .
  { Allow user to evaluate polynomials using usual notation: f(x). }
  return Evaluate(f, x);
end intrinsic;

map   := func< f, xs | [ f(x) : x in Eltseq(xs) ] >;
fst   := func< xs | xs[1] >;
snd   := func< xs | xs[2] >;

intrinsic CRT (T::SeqEnum[Tup]) -> .
  { Compute CRT from sequence of pairs <residue,modulus>. }
  return CRT(map(fst,T), map(snd,T));
end intrinsic;

intrinsic PohligHellman (g,h,p::RngIntElt,e::RngIntElt) -> RngIntElt
  { Compute discrete log x where g^x = h and g has order p^e. }
  return e gt 1 select x0 + p^e0*x1
      where x1 := recurse(g^(p^e0), h/g^x0,   p, e1)
      where x0 := recurse(g^(p^e1), h^(p^e1), p, e0)
      where e1 := e - e0
      where e0 := e div 2
      where recurse := $$
    else Rep({ d : d in [0..p-1] | g^d eq h });
end intrinsic;
```

```
intrinsic PohligHellman (g,h,n::RngIntElt) -> RngIntElt
  { Compute discrete log x from g^x = h where g has (hopefully smooth!) order n. }
  return CRT([ <PohligHellman(g^(n div p^e),h^(n div p^e),p,e), p^e>
                where p,e := Explode(f) : f in Factorisation(n) ]);
end intrinsic;

intrinsic PohligHellman (g,h) -> RngIntElt
  { Compute discrete log x from g^x = h where g has hopefully smooth order. }
  return PohligHellman(g,h,Order(g));
end intrinsic;
```

Some user code:

```
PSL27 := PermutationGroup< 8 | (2,3,5)(6,7,8), (1,2,4)(3,5,6) >;
S := MatrixAlgebra< FiniteField(2), 3 |
       [ 0,1,0,  1,1,1,  0,0,1 ], [ 1,1,1,  0,1,1,  0,1,0 ] >;
M := GModule(PSL27, S);
M: Maximal;

L := Lattice("E", 8);
S := ShortestVectors(L);
#S; // 120
KissingNumber(L); // 240
w := RSpace(RationalField(), 8) ! [ -1/6, 1/6, -1/2, -1/6, 1/6, -1/2, 1/6, -1/2 ];
C, d := ClosestVectors(L, w);
d; // 8/9
{ Norm(v): v in C }; // { 0, 2 }
{ Norm(v - w): v in C }; // { 8/9 }
```

Transcript of a session:

```
> P<x> := PolynomialRing(Rationals());
> f := x^3 + x + 1;
> Evaluate(f,1);
3
> f(1); // this doesn't work?! ridiculous!
>> f(1);
   ^
Runtime error in '@': Bad argument types
Argument types given: RngIntElt, RngUPolElt[FldRat]

> Attach("intrin.m");
> f(1);
3
> g := Integers(2^16384)!3;
> h := g^RandomBits(16384);
> time x := PohligHellman(g,h);
Time: 17.130
> g^x eq h;
true
> PohligHellman(g,g^1337) where g is Random(GL(3,127));
1337
```

3 Installation

There are at least 3 simple ways to use this "package" with the LaTeX `minted` package. In all cases, you have to run with the `-shell-escape` to LaTeX; *e.g.,*

```
bash$ pdflatex -shell-escape file.tex && xpdf file.pdf
```

3.1 Local copy of lexer

Place the `magma.py` file in the working directory and use

```
\begin{minted}{magma.py:Magma -x}
  // your magma code here
\end{minted}
```

3.2 Execute the lexer script

If you will be using a lot of `magma` code snippets, it is unpleasant to have to use the `magma.py:Magma -x` name over and over again. An alternative is to give the `magma.py` script execute permission, then in the preamble after loading `minted` do the following:

```
\renewcommand{\MintedPygmentize}{./magma.py}
```

Now your code segments can use the simpler `magma` format:

```
\begin{minted}{magma}
  // your magma code here
\end{minted}
```

3.3 Install globally

Details coming later.

References

[1] `https://pygments.org`.

[2] Wieb Bosma, John Cannon, and Catherine Playoust, The Magma algebra system. I. The user language, J. Symbolic Comput., 24 (1997), 235–265.

[3] `https://github.com/gpoore/minted`.

A Magma.py

Source code for the current version of `magma.py`.

```
#! /usr/bin/env python

import argparse
import sys
import pygments.cmdline as _cmdline

def main(args):
  parser = argparse.ArgumentParser()
  parser.add_argument('-l', dest='lexer', type=str)
  opts, rest = parser.parse_known_args(args[1:])
  if str(opts.lexer).lower() == 'magma':
    args = [__file__, '-l', __file__ + ':Magma', '-x', *rest]
  _cmdline.main(args)

if __name__ == '__main__':
  main(sys.argv)

# ----------------------------------------------------------------------
# -- here's the actual MagmaLexer:

from pygments.lexer import RegexLexer, include, words, bygroups, using, this
from pygments.token import Comment, Operator, Keyword, Name, String, Number,\
                           Punctuation, Whitespace, Text, Generic, Error

class Magma(RegexLexer):
  name = 'Magma'
  aliases = ['magma']
  filenames = ['*.m', '*.mag']

  # the following categories came from an Emacs Magma-mode package
  keywords =       """ _ by default do is select then to where end until
                       catch elif else when case for function if intrinsic
                       procedure repeat try while """
  operators =      """ adj and cat cmpeq cmpne diff div eq ge gt in join
                       le lt meet mod ne notadj notin notsubset or sdiff
                       subset xor not """
  extra_keywords = """ assert assert2 assert3 break clear continue declare delete
                       error error if eval exit
                       forward fprintf freeze iload import load local print
                       printf quit random read readi require requirege
                       requirerange restore return save time vprint vprintf vtime
                       assigned exists forall """
  constructors =   """ car case cop elt ext func hom ideal lideal map
                       ncl pmap proc quo rec recformat rideal sub """
  block_open =     """ try catch case when then else do repeat
                       function procedure intrinsic """
  block_close =    """ until end when elif else catch """

  keywords += operators + extra_keywords + constructors + block_open + block_close
  keywords = keywords.split()

  tokens = {

    'root': [
```

```
      include('whitespace-comments'),
      include('error-messages'),
      include('intrinsics'),
      include('number'),
      (r'"', String, 'string'),
      (r'\'[^\']+\'', Name.Builtin),
      (r'\.\.|::|:=|:->|->', Operator),
      (r'[^\w\s]', Punctuation), # do not add +
      (words(keywords, prefix=r'\b', suffix=r'\b'), Keyword.Reserved),
      (r'true|false', Name.Constant),
      (r'[A-Z]+[a-z]+([A-Z]+[a-z]*)?', Name.Builtin),
      (r'\w+', Name),
    ],

    'string': [
      (r'[^\\"]+', String),
      (r'"', String, '#pop'),
      (r'\\.', String.Escape),
    ],

    'whitespace-comments': [
      (r'\s+', Whitespace),
      (r'//.*', Comment),
      (r'/\*', Comment, 'comment-multiline'),
    ],

    'comment-multiline': [
      (r'\*+/', Comment, '#pop'),
      (r'\*+(?!/)', Comment),
      (r'[^*]+', Comment),
    ],

    'number': [
      (r'0[xX][a-fA-F0-9]+', Number.Hex),
      (r'0[bB][01]+', Number.Bin),
      (r'[0-9]+\.([0-9]+|(?!\.))', Number.Float),
      (r'[0-9]+', Number.Integer),
    ],

    'intrinsics': [
      (r'''(?sx)
        (intrinsic) (\s+)
        (\S+?) (\s*)
        ([(].*?[)].*?)
        (-> \s* (?: \S[^,]*? (?: \s* , \s* \S[^,]*?)* \s*))
        (\{)
      ''', bygroups(Keyword, Whitespace, Name.Function, Whitespace,
                    using(this), using(this), Comment), 'comment-intrinsic'),
    ],

    'comment-intrinsic': [
      (r'\{', Comment, '#push'),
      (r'\}', Comment, '#pop'),
      (r'[^\{\}]+', Comment),
    ],
```

```
    # This is likely an incomplete list, but who's going to typeset
    # transcripts with errors anyway?
    'error-messages': [
      (r'Runtime error in .*?:.*', Generic.Error),
      (r'Argument types given:', Generic.Error),
      (r'User error:.*', Generic.Error),
      (r'Magma: Internal error', Generic.Error),
      (r'Machine type: \S+', Generic.Error),
      (r'Initial seed: \S+', Generic.Error),
      (r'Time to this point: \S+', Generic.Error),
      (r'Memory usage: \S+?MB', Generic.Error),
      (r'Internal error at \S+, line \S+', Generic.Error),
      (r'Illegal operation', Generic.Error),
      (r'Time:', Generic.Error), # HACK
    ],

  }

# -*- mode: Python; python-indent-offset: 2; python-guess-indent: nil -*-
```

B Tests

Here are some tests based on code extracted from various Magma packages.

```
intrinsic Conjugates(G::Grp, H::Grp: Limit := 10000000) -> {}
  {The set of conjugates of H by elements of G}
  require Type(G) eq Type(H) and H subset G:
          "Argument 2 is not a subgroup of argument 1";
  N := Normalizer(G, H);
  require Index(G, N) le Limit:
    "Number of conjugates of H in G is more than " cat Sprint(Limit);
  if Type(G) eq GrpPC then
    return { PowerGroup(G) | H^t: t in Transversal(G, N) };
  end if;
  return {H^t: t in Transversal(G, N)};
end intrinsic;

/* Given an ideal corresponding to an absolutely irreducible trace tuple t and hence
 * an absolutely irreducible representation Δ: F_2 → SL(3, K), find a tuple of
 * nine words (w_1, ..., w_9) such that (Δ(w_1), ..., Δ(w_9)) is a basis of K^{3×3}.
 *
 * Note that (Δ(w_1), ..., Δ(w_9)) is a basis if and only if the matrix
 * (tr(Δ(w_i)Δ(w_j)))_{i,j} has full rank, hence non-zero determinant.
 */
L3FindBasis := function(I)
  if exists(b){ b : b in possibleBases | GramMatrixDeterminant(b, I) ne 0 } then
    return b;
  end if;
  error "not absolutely irreducible";
end function;

intrinsic IsInTwistedForm( x::GrpLieElt, c::OneCoC ) -> BoolElt
```

```
  {Returns true iff x ∈ G(K) is an element of the twisted group of Lie type G_c(k)}
  G := Parent(x);
  require G cmpeq Domain(Group(GammaGroup(c)))
       : "Parent(x) and the cocycle do not match";
  Gamma, m := ActingGroup(GammaGroup(c));
  return forall{ i : i in [1..Ngens(Gamma)] |
                 x eq x @ FieldAutomorphism(G, m(Gamma.i)) @ c(Gamma.i) };
end intrinsic;

intrinsic Bezoutian(H::HypGeomData) -> RngIntElt // e.g. [1,2,3],[5] -> 5
  {The resultant of the defining polynomials of the hypergeometric data}
  f,g:=DefiningPolynomials(H); return Resultant(f,g); end intrinsic;
```

Here are some examples for testing edge cases.

```
intrinsic myintrin1(G :: GrpMat[FldFin]) -> {}, [], {RngIntElt}
  { some documentation }
  return {}, [], 0; end intrinsic

intrinsic myintrin2(x::{}, y::{[]}) -> {}
  { some documentation }
  return {}; end intrinsic

intrinsic myintrin3(G:: GrpMat : Degree := "All", Set := {}) -> BoolElt, {}
  { { { { some } more } documentation { with { [ crazy ] braces } } } }
  return false; end intrinsic;
```

Here are examples of valid Magma code that fail to format correctly; these examples are currently considered too extreme to worry about.

```
// incorrectly matches braces within /* */ comment
intrinsic junk(a) /* {} */ { docs }
  return a+1; end intrinsic;

// can't have comment between intrinsic name and open paren
intrinsic junk /* */ (a) /* comment
  */ { doc } printf "a = %o\n", a; end intrinsic;
```